RFC: Lightbike protocol

Owner: Caleb Boylan

Foreword

This RFC targets video game developers to create a standard for the networking protocol that drives the multiplayer Lightbike game. Any video game developers that wish to create a Lightbike game should contribute to this document in order to create a good and useful protocol.

The Lightbike protocol aims to create a standard protocol for networked Lightbike games. The goal of the protocol is to be sturdy and simple, but it must also be quick, so it will use UDP and not worry that some data may be lost in transit.

Motivation

The Lightbike protocol is important to create a strong standard so any Lightbike client and server can communicate with each other seamlessly. It also needs to be a fast and simple protocol in order to be able to transfer data to and from the server so the clients can display each new update as fast as possible. Because the standard defined is the network protocol, the manner in which the client displays the game is up to the client, in some cases it may not even provide a graphical interface, this would be the use case of creating a bot to play the game. This also makes it easy for someone to create a client that fits their needs or choose a client that they think is best.

Background

The Lightbike game is a game in which there are 2 to 4 players, that travel across a board of some size defined at the beginning of the game. As each player travels, they leave a tail behind them and if a player runs into the tail, they die. The last player alive wins the game. The Lightbike game can support more than 2 characters, but a board of the appropriate size must be used based on the number of players wanting to play. The network requirements also grow as the number of players grows.

Requirements

The requirements for this RFC to be considered a success are for a complete protocol to be defined and for the network requirements to be relatively small so that 2 players with low latency should be able to play with each other without problem.

The RFC creates no standard for how the clients work on the frontend. It only creates a standard for how the data must be sent over the network to and from the server.

By the end of this RFC it should be clear how the network protocol works and how you can implement your own server or client for the Lightbike game.

Implementation

The Lightbike protocol has several different pieces that are explained in the following sections. The Lightbike protocol works entirely over UDP so it must work around the limitations of UDP.

Game Creation

Before users can authenticate a game must be created:

Client:

```
CREATE <number of players>
```

Server:

```
CREATE SUCCESS
```

Failed creation looks as follows:

```
CREATE ERROR 1
```

This occurs when a game has already been created and is still running.

```
CREATE ERROR 2
```

This occurs when the number of players is lower than 2 or greater than 4. Two people are required to play the game therefore a game of size 1 does not make sense and cannot be created.

Authentication

The authentication of the Lightbike protocol is simple. First the client sends the server a message saying it wishes to connect, then the player’s name, the server then responds with a token for the client to use. This is important as UDP is connectionless, so there is no way to associate a user with a specific connection. An example of what the authentication handshake looks like is as follows.

Client:

```
AUTH calebb
```

Server:

```
AUTHACK calebb <TOKEN>
```

Fail messages:

```
AUTHFAIL 1
```

This indicates that the game is full.

```
AUTHFAIL 2
```

This indicates that the username the user is trying to use has been taken already.

TOKEN is a string, the generation of this string is not defined by this RFC, but it should be unique. It could be a random string of characters, or it could be a hash of the player's username and a salt. It should not be a predictable string.

The server will then respond to the client using the IP Address and Port that the Authentication packet comes from.

Game Start

The game is started when the server send the first UPDATE to each user, see below for information of how that works.

Game Data

Every frame of the game, the server will send the current game board to each user, and the user will be able to update the server with what actions they wish to take. The game will send the game board as a string of characters, with each row of the board being separated by a newline. The valid characters are:

P: Player location
E: Enemy location
0: Empty block
1: Block that has been traveled over

The message sent by the server looks as follows:

```
UPDATE <GAME BOARD>
```

The client can send updates to the server as well, this message looks as follows:

```
DIRECTION <TOKEN> <DIRECTION>
```

TOKEN is the token that the server gave the client during the authentication step.

DIRECTION is the new direction that the player is facing. Obviously the character cannot turn 180 degrees, if the client sends a direction that is 180 degrees from their current direction, the server will ignore it.

Game End

The last player alive is the winner. The server will notify the clients of the winner as follows:

```
WINNER <WINNER_NAMES>
```

WINNER_NAMES is a space separated list of winners. In the case of a tie this will be more than one name, otherwise it is a single name.

After the client receives this message the game is over and it must authenticate to the server again if it wishes to play again.

Appendix

Client: A program that receives input from the user and sends it to the server, as well as receives data from the server and displays it for the user

Server: A program that takes data from the client, applies game logic to it, and updates the clients of the new gamestate

Token: A string used to identify a user as an authenticated user.